Tcl and octal numbers

Tcl mathematical operation commands treat strings begininning with 0 as octal numbers, catching the unsuspecting programmer off-guard.

AMG: Not in Tcl 9 [L1]! Tcl 9 octal numbers must begin with 0o. Tcl 8.6 supports (but does not require) this notation.

See also

Frequently Made Mistakes™ in Tcl : Zero, Cameron Laird

Description

The standard tcl::mathop mathematical operation commands treat strings beginning with 0 as octal numbers. A basic example:

```
::tcl::mathop::+ 010 010 ;# -> 16
expr {010 + 010} ;# -> 16
```

This behaviour can conflict with the practive of padding decimal numbers with leading zeroes, since in standard Tcl numerical contexts, such decimal numbers are then treated as octal numbers, leading to unexpected results. Here's an example of the gotcha:

```
set hours    08
set minutes  45
set seconds  00
set newhours [expr {$hours + 1}]
```

The result:

```
can't use invalid octal number as operand of "+"
```

08 is not a valid octal number, and expr won't simply interpret it as an decimal 8 that's left-padded with a 0.

To work with zero-padded decimal numbers, use scan, telling it to treate numbers with leading zero's as decimal numbers:

```
scan $hours %d hours
```

This strips off the leading zeros. It's also safer than

```
string trimleft $hours 0
```

, which can fail if for example $hours ever ends up being 00, or if $hours is negative (not likely in a clock context, but the argument still applies).

Lars H 2008-07-04: One problem with the above that turned up when doing arithmetic on currency is that %d doesn't handle arbitrarily-large integers (even though Tcl can) — it's restricted to the range of numbers int() can output. In order to handle integers with an arbitrary number of digits, it is necessary to do

```
scan $hours %lld hours
```

Obviously this is not an issue with a number of hours in a day, but it *can* be an issue for other numbers.

---

glennj: one potential pitfall of scan is that it might mask potential errors. The following example fails as expected:

```
set n 09blah42
incr n

expected integer but got "09blah42"
while evaluating {incr n}
```

However:

```
set n 09blah42
scan $n %d n
incr n ;# ==> n is now 10
```

Application writers might actually want to trap an invalid entry like that.

AMW 2012-12-23: I use the following regular expression in my code to prevent accidental interpretation as octal:

```
# assure that $num is not interpreted as octal:
regexp {^0*(\d+)} $num _dummy num
# 0 -> 0
# 0000 -> 0
# 0123 -> 123
# notanumber -> notanumber
# 0123dollar -> 123
```

If you want to avoid the interpretation of the last example as a number, add a final $ to the regexp:

```
regexp {^0*(\d+)$} $num _dummy num
# 0 -> 0
# 0000 -> 0
# 0123 -> 123
# notanumber -> notanumber
# 0123dollar -> 123dollar
```

AMG: I suggest [scan %d]. This also forces decimal interpretation. scan 0123 %d returns 123.

Donald Arseneau 2003-03-12: I see Kevin Kenny contributed the following to c.l.t

```
proc forceInteger { x } {
    set count [scan $x %d%s n rest]
    if { $count <= 0 || ( $count == 2 && ![string is space $rest] ) } {
        return -code error "not an integer: \"$x\""
    }
    return $n
}

% forceInteger x
not an integer: "x"
% forceInteger 123
123
% forceInteger 08
8
```

This also covers my preceding concern:

```
% forceInteger 09blah42
not an integer: "09blah42"
```

Better than an explicit test of

```
string is space $rest
```

is to just skip (optional) spaces in the scan pattern:

```
proc forceInteger { x } {
    set count [scan $x {%d %c} n c]
    if { $count != 1 } {
        return -code error "not an integer: \"$x\""
    }
    return $n
}
```

[Explain improved diagnostic in 8.3.]

The question recently came up *how do I display the octal value of a character in Tcl*? and according to RS, the complete sequence is

```
format 0%o [scan a %c]
```

(but only with Tcl more recent than 8.2 or so; older scan works slightly differently).

Non-arithmetic Use of expr

RS 2006-06-19 PYK 2015-05-30: A word of warning: expr treats any value that looks like a number as a number, even if no arithmetic operation is performed on that value:

```
% set bond james
% expr {$bond eq {} ? {-} : $bond}
james
% set bond 0070
% expr {$bond eq {} ? {-} : $bond}
56
```

If a value cannot be treated as a number, perhaps because it is invalid as an octal number, it is left as a string value:

```
% set bond 008
% expr {$bond eq {} ? {-} : $bond}
008
```

In such cases it's better and more robust to use if:

```
if {$bond eq {}} {set bond -}
```

HE 2006-06-20: Strange behavior! The manpage of expr says:

```
 eq ne
  Boolean string equal and string not equal.
  Each operator produces a zero/one result.
  The operand types are interpreted only as strings.
```

There is no mention about this behavior. (I remember weakly this two operators are added exactly to avoid this problem) More interesting: The manpage of if says:

```
 The if command evaluates expr1 as an expression (in the same way that expr
evaluates its argument).
```

Is there something wrong?

JMN 2006-10-19 PYK 2015-05-30: Not really.. The value being treated as a number is the second value in the ternary operator. This may make it clearer:

```
% expr {$bond eq {} ? {-}: "hello $bond"}
hello 0070
% expr {$bond}
56
% expr {$bond eq 56}
0
% expr {$bond == 56}
1
```

LV: Note the previous discussions on this page regarding the precautions one should keep in mind when using eq on tcl variables which contain numeric values. I am not certain I can think of a case where one would use eq when comparing numeric values...

string is ... Does Not Understand Octal

```
string is integer 098 ;# -> 0
```

IDG: There appears to be an octal related bug in string is. string is double 098 returns 1. (8.4.1 on windoze)

PT: At the very least it's inconsistent (tcl 8.5a0 win98)

```
% string is integer 098
0
% string is double 098
1
```

PYK 2015-05-30: string is integer returns 0 for 098 not because it doesn't understand octal but because it **does** understand octal, and 098 is not a valid octal number. Also, in recent versions of Tcl, both string is integer 098 and string is double 098 return 0.

TIP #114

TIP 114 proposes modifying Tcl in a future release so that numbers beginning with 0 are *not* treated as octal numbers. The proposer believes that far more users stumble upon this feature by accident than use it intentionally.

PYK 2015-05-30: As has already been mentioned at the beginning of this article, TIP 114 is implemented in Tcl 9.

Misc

2003-12-22 VI: What I'd like more than what TIP 114 specifies is a prefix like 0d, which would force the rest of the number to be interpreted as decimal.

For the specific clock case, where we know we have two digits, I like to use expr like this:

```
set m [clock format [clock seconds] -format %m]
set m [expr 1$m % 100]
```

LV: Right now, when I say:

```
set abc \1
```

abc is set to an octal 001.

Once this TIP is implemented, what will happen to the above code? Is it going to change behavior? - RS: *abc* is set to a string of one character U+0001 (ASCII SOH) - unaware of decimal, hex, or octal. This page is about parsing integers from strings, and U+0001 or any non-digit cannot be parsed as integer anyway :^)

Chris Nelson points out that octal interpretation can happen in Javascript as well. See comp.lang.javascript FAQ .

---

snichols 2007-02-26: I tested octal arithmetic in both Ruby's irb interpreter and Python's interpreter and they behave in a similar way. So, this seems to be a common pitfall with other scripting languages too:

Ruby's IRB

```
irb(main):001:0> 09 + 100
SyntaxError: compile error
(irb):1: Illegal octal digit
09 + 100
 ^
```

Python

```
>>> 09 + 100
 File "<stdin>", line 1
   09 + 100
    ^
SyntaxError: invalid token
```